NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

PROVIDING CRYPTOGRAPHIC SECURITY AND EVIDENTIARY CHAIN-OF-CUSTODY WITH THE ADVANCED FORENSIC FORMAT, LIBRARY, AND TOOLS
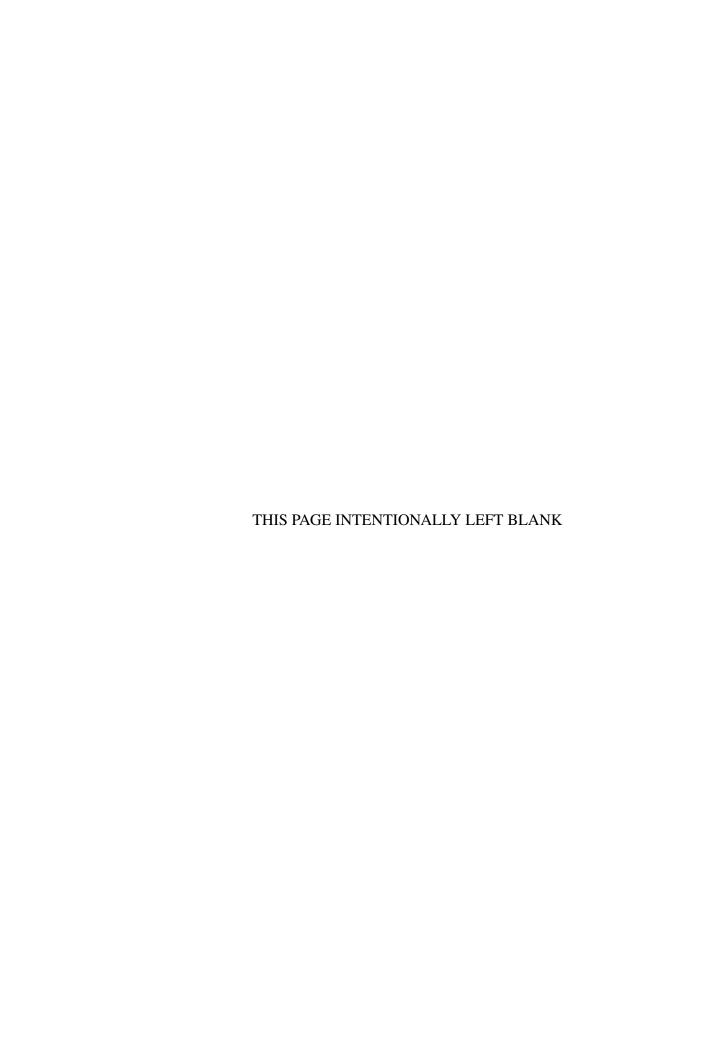
by

Simson L. Garfinkel

August 19, 2008

THIS PAGE INTENTIONALLY LEFT BLANK

NAVAL POSTGRADUATE SCHOOL
Monterey, California 93943-5000

Daniel T. Oliver
President

Leonard A. Ferrari
Executive Vice President and Provost

Approved for public release; distribution is unlimited

This report was prepared by:

Simson L. Garfinkel
Associate Professor

Reviewed by:

Released by:

_____
Peter Denning
Department of Computer Science

_____
Dan C. Boger
Interim Vice President and
Dean of Research

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704–0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202–4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
19–8–2008	Technical Report	2007-09-01—2008-07-31

4. TITLE AND SUBTITLE

Providing Cryptographic Security and Evidentiary Chain-of-Custody with the Advanced Forensic Format, Library, and Tools

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Simson L. Garfinkel

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Postgraduate School

8. PERFORMING ORGANIZATION REPORT NUMBER

NPS-CS-08-014

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited

13. SUPPLEMENTARY NOTES

The views expressed in this report are those of the author and do not necessarily reflect the official policy or position of the Department of Defense or the U.S. Government.

14. ABSTRACT

This paper presents improvements in the Advanced Forensics Format Library version 3 that provide for digital signatures and other cryptographic protections for digital evidence, allowing an investigator to establish a reliable chain-of-custody for electronic evidence from the crime scene to the court room. No other system for handling and storing electronic evidence currently provides such capabilities. This paper discusses implementation details, user level commands, and the AFFLIB programmer's API.

15. SUBJECT TERMS

AFF, AFFLIB, AImage, Advanced Disk Imager, disk imaging, EnCase

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
Unclassified	Unclassified	Unclassified	UU	46	19b. TELEPHONE NUMBER *(include area code)*

THIS PAGE INTENTIONALLY LEFT BLANK

1   Introduction

Chain-of-custody for evidence from the crime scene to the court room is a bedrock principle of both civil and criminal law. Without a clear and unambiguous chain-of-custody there is no way to be sure that an object presented to the court is the same object that was collected at the scene of the crime. Even evidence presented to technical experts needs to have chain-of-custody: without it, there is no way to assure that the expert's testimony pertains to evidence from the actual case that is under consideration.

A paper notebook found at a crime scene can be put into an evidence bag, tagged, and locked away in an evidence locker. Each time the evidence is accessed or moved to another location this fact will be noted. In this manner the prosecution can show that the evidence has not been tampered; in the rare cases where tampering takes place, it can be detected.

But unlike records written with pen and paper, digital files can be modified without leaving a trace of the original message. This is one of the great challenges of digital forensics—establishing that a particular arrangement of bits on a digital storage medium is the result of on specific computational history (*e.g.,* deleting a file) and not of another (*e.g.,* using a hex editor to write raw sectors onto the disk drive that are indicative of a deleted file) [Carrier, 2006].

Of course, hard drives, USB memory sticks, and cell phones are tagged and bagged. But at some point, the information on these devices needs to be copied onto another computer system for analysis. In a modern forensic laboratory these files might be placed on a high-capacity server or a Storage Area Network (SAN) to allow for flexible use and simultaneous access by multiple examiners. Such environments require highly reliable technical measures to provide assurances that evidence is kept intact and unmodified.

Although computer forensics practitioners understand the importance of chain-of-custody, today's tools for preserving this chain are poor. Programs such as EnCase [Keightley, 2003] and dcfldd [Harbour, 2006] will compute an MD5 or SHA-1 cryptographic hash of a disk when it is copied by an investigator into an *image file*. Later, when the image file is provided to a forensic analyst, the analyst can compare the hash of the image received with the hash of the original to determine if the file has been modified. If the hashes match, the assumption is that the file is unchanged from the original.

This paper introduces an improved method for assuring the integrity of digital evidence that is based on public key cryptography. In addition to providing improved integrity, the method presented also allows for:

- digital documentation of evidentiary transfer from one agent to another;

- reconstruction of evidence that has been inadvertently damaged during transfer;

- forensically sound methods for recovering partial evidence in cases where so much digital evidence has been damaged that reconstruction is no longer possible;

- encryption with both symmetric and public key cryptography, so that evidence that is acquired in a hostile environment can be safely transferred back to a secure facility.

These new methods have been implemented in the Advanced Forensic Format Library (AF-FLIB) Version 3 [Garfinkel, 2008]. AFFLIB is an open source software package written in the C/C++ programming language that allows for the imaging, manipulation, storage and use of digital evidence. The software is available free of charge for incorporation into both open source and proprietary forensic applications.

2   Background and Prior Work

2.1   Disks and Disk Images

Computer hard drives, optical drives, and solid state drives are mass storage devices that organize the information they store as a series of numbered, fix-sized *sectors*. Traditionally hard drives employ a sector size of 512 bytes and CDROM drives used 2048-byte sectors, although a standard for 4096-byte sectors is currently under development [Fonseca, 2007].

A *disk image file*, or more generally an *image file*, is a file that contains a sector-for-sector copy of the contents of a mass storage device. Image files are intended to be perfect copies of the disk's contents. Image files are produced with programs called *imagers*.[1]

Although the discussion to this point has focused on disk image files, in practice any data carrying device can be imaged. Once a device is imaged, the forensic investigator works with the image, rather than the original device, in order to preserve the device's integrity: most computer forensic tools can directly read and process disk image files.

Image files are particularly important when it is necessary to record the state of a device that must then be returned to service—for example, in the event of a network attack. In these cases, the image file may be the only tangible evidence of the crime that has taken place after the system has been restored to operation.

Imaging also provides a simple and operating system independent means for backing up a hard drive: the drive is simply imaged into a file or onto another drive. To restore the backup, the image is *restored* on the original drive. The image can also be restored on another drive of similar size, a process sometimes called *cloning*.

Disk image files can be stored in different formats. The most basic format is the *raw* format in which the bytes in the image file have a one-for-one correspondence to the bytes on the physical device (*e.g.,* bytes 0–511 in the file represent the first 512-byte sector, bytes 512–1023 represent the second 512-byte sector, and so on). The advantage of the raw format is that it is easy to understand and easy to implement; the disadvantage is that raw files consume as much storage

---

[1]A comprehensive list of disk imagers can be found on the Forensics Wiki at `http://www.forensicswiki.org/index.php?title=Category:Disk_imaging`.

space as the device being imaged: imaging a newly purchased 80GB hard drive will produce an 80GB raw file, even though each one of the drive's 160 million 512-byte disk sectors is filled with ASCII NULLs.

2.2 Disk Image Formats

There are two important shortcomings that forensic examiners experience when working with raw disk images: the images are unwieldy, and they do not capture important information such as the time that the disk was imaged, who performed the imaging, or even the device sector size. Because of these shortcomings, developers have created a number of disk image formats, each with its own intended purpose.

One of the most widely used file formats today is the EnCase Evidence File Format. This format is based on the ASR Data Expert Witness Compression Format [ASR, 2002]. Disk images are stored as a series of files, each file not exceeding 2GB ($2^{31} - 1 = 2,147,483,647$ bytes). The first file contains a "Case Info" header, a table containing a 32-bit CRC and the offset of each "blocks" in the disk image (the default block size is 64 sectors), and a footer containing an MD5 hash for the entire physical disk. Also contained in the header are the date and time of acquisition, an examiner's name, notes on the acquisition, and an optional password; the header concludes with its own CRC. Images that require more than 2GB of storage are split into multiple files and given file names such as `FILE.E01`, `FILE.E02`, *etc.* Disk images can be split into files smaller than 2GB for storage to archival CDROM. The EnCase/Expert Witness file format can be read by a number of commercial programs and by the Open Source Libewf [Kloet et al., 2008].

Other forensic file formats include a proprietary format used by AccessData's Forensic Toolkit (FTK), the file format used by Safeback [NTI Forensics Source, 2008], and the file format used by ILook Investigator [US Treasury, 2008]. A detailed survey of forensic file formats appears in [Garfinkel et al., 2006].

In almost all cases it is faster to perform a forensic investigation with a an uncompressed raw file than it is to work with a compressed file. This is because modern forensic programs frequently need to skip from one part of a disk image to another: when a compressed format is used, parts of the disk image are constantly being read off the disk, decompressed, and then discarded. Using an uncompressed format avoids the decompression step, which is computationally intensive.

2.3 Assuring Integrity with Hash Functions

Forensic practitioners today largely rely on the MD5 [Rivest, 1992] and SHA-1 [Computer Systems Laboratory and Technology, 1993] cryptographic hash functions to assure the integrity of images that they acquire.

A cryptographic hash is a one-way function which takes an arbitrary amount of input and pro-

duces an output of a fixed size. (Cryptographers will sometimes call the input a *pre-image* and refer to the hash value as the *residue*.) To be considered strong, a cryptographic hash function should have these properties:

- **Preimage resistance**: Given any hash, it should be computationally infeasible to find a specific preimage that produces the residue.

- **Second preimage resistance**: Given a message m_1, it should be computationally infeasible to find a message m_2 that has the same hash.

- **Collision resistance**: It should be computationally infeasible to find *any* two messages m_1 and m_2 that have the same hash. [Friedl, 2005, Boneh, 2001]

The MD5 algorithm produces a 128-bit cryptographic hash; this hash is typically written as 32 hexadecimal digits. The SHA-1 algorithm produces a 160-bit hash which is typically written as 40 digits.

Today it is common practice for computer forensic investigators to record the MD5 or SHA-1 of a disk when it is imaged. The hash is computed by the acquisition tool as the data is read from disk being imaged and displayed on the computer's screen; the investigator records this number in the investigative report.

For example, in the case of US v. Zacarias Moussaoui, when the contents of Moussaoui's laptop were duplicated by the FBI with Safeback, a program was used to compute the MD5 of both the laptop's drive and the copy made by the FBI. A copy of the laptop's drive was then provided to Mr. Moussaoui's defense team. The MD5 of this copy was computed and compared with the MD5 of the original laptop's drive. According to court filings:

> "The significance of this point is two-fold. First, there can be no question that the defense has the exact same copy of the original that the Government has, so they can conduct any further investigation on their copy that they wish. Second, the results of the MD5 program as to these two laptops further demonstrate the reliability of the Safeback program." [Novak, 2002]

There are several advantages to the current practice of manually recording hash codes and incorporating them into investigative reports:

- The practice is easy to understand.

- The practice is in general use.

- The practice is easy to explain in court.

- The hash codes are easily recorded in an investigative report which the investigator is presumably already keeping for other purposes.

- The same procedures which assure for integrity of the investigative report will similarly assure for the integrity of the hash codes.

But today's practice has potential problems as well:

- Because the hash codes are recorded in what is essentially a free-format report narrative, it is difficult to apply automated processing and validation.

- If the disk image becomes corrupted, the hash code will only report that it no longer matches: it does not allow the error to be isolated or corrected.

2.4 MD5 Vulnerabilities

In recent years a number of vulnerabilities have been found in the MD5 hashing algorithm, culminating with the discovery of MD5 collisions [Wang et al., 2004]. For this reason MD5 is no longer considered by computer scientists to be a good choice for security-critical applications. Although as of this writing no SHA-1 collisions have been publicly announced, many researchers feel that it is only a matter of time [Schneier, 2004]. Increasingly security software uses the SHA-256 algorithm, which produces a 256-bit hash, and NIST has started an effort to develop a new hash standard [NIST, 2007].

There are at least two reasons that the discovery of MD5 collisions was not as catastrophic for computer security in practice as they might otherwise have been:

- First, although it is possible to generate MD5 collisions, it still takes a considerable amount of computer power and expertise to do so.

- A second and perhaps more important reason is that modern security engineering practice is to use a plug-in architecture for cryptographic algorithms. To be "pluggable" formats must store version numbers, algorithm names and key lengths in data that transmitted or stored. The practical result of this engineering practice is that most programs that employ hash functions can work with a wide range of algorithm. Software that relies on cryptographic hashes can then validate using any or all of these algorithms, dramatically reducing the chances that an attack will be successful.

2.5 Piecewise Hashing

In addition to computing a hash of the entire disk image, some tools will compute a hash for individual sections or "pieces" of the image. For example, `dcfldd` [Harbour, 2006] can compute a hash for each block and store the hashes in a separate file. This approach of separately hashing each piece of the file is called *piecewise hashing*.

Piecewise hashing is an important advance for digital forensics. Whereas a single hash code for an image can establish that an image has not been modified, if the file is modified the piecewise hashes can be used to help determine the location and extent of the alternation. Changing a single bit from a 0 to a 1 will change the hash for the entire image, but it will only change one of the piecewise hashes. In such a case, the remaining pieces would still have evidentiary value. Even if a file is truncated—for example, an 80GB file cut into a 20GB file—the piecewise hashes will allow the remaining evidence to be used, as long as it is otherwise unaltered.

2.6 Digital Signatures for Data Integrity

Digital signatures were invented by Diffie and Hellman for the purpose of securing mail sent over digital networks such as the Internet [Diffie and Hellman, 1976]. Digital signatures in the form of digital certificates have been applied for the purpose of certifying cryptographic keys [Kohnfelder, 1978], and now provide authentication for the vast majority of encrypted communications on the Internet through their incorporation into the SSL and TLS protocols [Dierks and Allen, 1999]. Digital signatures have also been widely applied to code signing in the Windows and Macintosh operating systems, as well as signing Linux software updates. But prior to the work presented in this paper, digital signatures have not been applied to imaging of digital media for forensic purposes.

Modern digital signatures are implemented as functional compositions of cryptographic hash functions and public key cryptography. In practice a document that is to be signed is hashed with a function such as SHA-1. The hash is then encrypted using an asymmetric encryption algorithm such as RSA [Rivest et al., 1977].

Asymmetric encryption algorithms have the property that data encrypted with an *encryption key* can only be decrypted with a matching *decryption key*. In practice one key is kept confidential (the *private key*) while the other key is disclosed (the *public key*). When used for digital signatures, the private key is used to sign the signature while the public key is used to validate to signature.

Verifying a digital signature accomplishes two purposes: it verifies that the digital document has not been modified, and it verifies that a particular private key was used to create the signature. Verification is typically performed in three steps. First, the document's hash function is computed for a second time. Next, the signature is decrypted with the signer's public key. Finally, the computed hash is compared with the decrypted hash: if they match, the signature verifies.

2.7 Hash Functions Alone are not Digital Signatures

It is important for forensic practitioners to understand that what gives the digital signature its security is the use of a private key to mathematically sign the cryptographic hash: a cryptographic hash by itself is not a digital signature.

This is an important distinction, because the terms "digital signature" and "forensic signature" are frequently—and incorrectly—used by forensics practitioners in reference to a simple cryptographic hash (see [Haggerty and Taylor, 2007, ICS, 2008]). A hash value by itself is not a signature, because it is not based on any secret information: anyone in possession of the data can generate the hash; thus, having the hash is not proof that a specific person or system had posession of the data.

True digital signatures are important for establishing chain-of-custody because of their *non-repudiability* properties. If the signer's private key has not been compromised and if the signature is valid, then the private key was used to create the signature[2]. One can easily imagine a future in which digital evidence is routinely signed using trusted hardware such as a US Department of Defense Common Access Card [US Department of Defense, 2008]. Such a signature provides not such an assurance that the evidence has not been tampered—it provides an electronic proof that a specific person (or, at least, a specific CAC) was used to sign the evidence *when* it was acquired. Other information such as GPS coordinates or a secure timestamp [Adams et al., 2001] could be included in the signature as well.

3   AFF and AFFLIB 3

The Advanced Forensic Format (AFF) is a format for storing digital evidence and associated metadata. Similar to the Expert Witness Format, AFF stores digital information as a series of blocks, range in size from 512 bytes to 4GB, which can be optionally compressed and stored in one or more disk files. Unlike Expert Witness, AFF is an extensible format which can store any kind of arbitrary data or metadata. To this end, AFF can be thought of as two parts: a container file format, similar to the ZIP file format, and a schema for mapping digital evidence to specific name/value pairs. A detailed description of the disk representation for the AFF format has been previously published [Garfinkel et al., 2006].

AFFLIB™ is an implementation of AFF written in a portable C++ that can be called from either C or C++. Rather than forcing the programmer to understand segments, data segments, compression and so on, AFFLIB implements a simple abstraction that makes the AFF image file appear as two resources: a simple name/value database that can be accessed with traditional `put` and `get` semantics; and a stream that can be accessed using `af_open()`, `af_read()`, and `af_seek()` function calls. If `af_open()` is used to open a non-AFF file, the library defaults to a pass-through mode of operation, allowing AFF-aware programs to work equally well with raw files. In this manner, it is easy to modify existing forensics software to work with AFF yet retain compatibility with raw files.

3.1   AFF Design

Specific goals for AFF are presented in [Garfinkel et al., 2006] and repeated in Figure 1. AFF accomplishes these goals by partitioning the format into two layers: a **data storage layer**, which

---

[2]Assuming that the signature algorithm itself has not been compromised, of course

- Ability to store:

  - disk images with or without compression.
  - disk images of any size.
  - metadata within disk images or separately.
  - images in a single file of any size or split among multiple files.
  - Arbitrary metadata in the form of user-defined name/value pairs.

- Extensibility.
- Simple design.
- Multi-platform, open source implementation.
- Freedom from any intellectual property restrictions.
- Provisions for internal self-consistency checking, so that part of an image could be recovered even if other parts of the image were rendered corrupt or otherwise lost.
- Provisions for certifying the authenticity of evidence files both with traditional hash functions like MD5 and SHA-1 and with advanced digital signatures based on X.509(v)3 certificates.

Figure 1: AFF Design Goals, from [Garfinkel et al., 2006]

specifies how the named AFF segments are stored in an actual file, and a **data schema layer**, which defines how the information stored in the named segments is to be interpreted.

3.2   AFF Data Storage Layer

The AFF data storage layer stores any number of name/value pairs within a single AFF object. AFF calls these name/value pairs *segments*. The segment name consists of a Unicode string between 1 and 64 characters long; the value consists of a 32-bit unsigned integer and a sequence of between 0 and $2^{32} - 1$ bytes. As discussed below in Section 3.3, different names are used to store different kinds of data and metadata.

When AFF is used to store disk images, the `pagesize` segment stores the size of each section of the disk image, the name `page0` is used to store the first section, `page1` is used to store the second, and so on. As the name implies, these sections of the disk image are called *pages*. By default AFF uses pages that are 16MB (2^{24}) bytes in length, although this can be changed on a file-by-file basis when the image file is created.

AFFLIB Version 3 was released in the fall of 2007 and has been steadily improved since. AFFLIB 3 includes supports for five different data storage layers:

8

AFF: A disk image in a single file. The AFF file format stores AFF segments in a single file that consists of a file header, one or more AFF segments, and a file footer. The format is designed to allow easy parsing and validation of AFF files and easy data recovery in the event of media failure.

Unlike the Expert Witness format, the AFF format store an entire disk image and associated metadata in a single file. This is designed to aid processing and ease-of-use in environments that work with dozens or even thousands of drives simultaneously. As a result, we could not use an existing archive format such as ZIP or JAR because neither supported files larger than 4GB due to the use of 32-bit offsets within the file directory. Likewise, we couldn't use compressed tar files because they do not provide for random access. In retrospect we could have used the ZIP64 format, but at the time we did not have an implementation of ZIP64 that was both clean and Open Source.

When an application asks AFFLIB to open an AFF file with the `af_open()` call, AFFLIB scans the entire file, noting the offset of each segment, and builds an in-memory table of contents with this information. Offsets stored within each segment allows the file to be read quickly, without necessitating the reading of each byte in the file. Although it would be possible to store the table of contents at the end of the file, the way the ZIP file format does, we decided to force a trip through the segment headers within the file as a way of quickly verifying the file's integrity. Offsets stored within the file allow reading only the segment headers, rather than forcing a read of the entire file contents. In practice this process takes between 10 and 30 seconds on a modern desktop system for image files of devices ranging from 10GB to 200GB. Modern operating systems cache disk sectors, so once a file is opened, subsequent file openings are nearly instantaneous as long as these sectors remain in the host operating system's cache.

AFD: Multiple AFF files in a single directory. Despite the fact that the AFF file format supports files larger than 4GB, some file systems (*e.g.,* MSDOS) do not support files larger than 2GB. On these systems AFFLIB supports an alternative storage mode called AFD, in which multiple AFF files are stored in a single directory. When a directory ending in the extension `.afd` is passed to AFFLIB's `af_open()` routine, AFFLIB scans the directory for `.aff` files and builds a single table of contents for all of the files. The maximum size of each AFF file within the AFD directory can be specified as an option.

AFM: Raw files with AFF annotations. A single AFF file can be used to store metadata or other annotations (for example, digital signatures) for a disk image that is stored in one or more raw files. In this case the file is given an AFM extension.

For example, if a disk image is stored in three files, `file.000`, `file.001` and `file.002`, annotations can be stored in a file called `file.afm`. Opening the `file.afm` file with AFFLIB causes the library to automatically locate and reference the data in the raw files when the forensic application attempts to read file data.

RAW and Split-Raw: Support for raw files. The AFFLIB library can also directly open raw or split-raw files if their file names are passed to the `af_open()` call.

S3: Storing on Amazon's Simple Storage Service. For supporting grid computing applications using Amazon's EC2, AFFLIB has the ability to directly store disk images inside Amazon's Simple Storage Service [Amazon, 2008].

Libewf: Legacy support for EnCase. Finally, LIBAFF can directly read disk images created in the Guidance Software EnCase file format using libewf [Kloet et al., 2008].

The AFFLIB `af_open()` determines which storage layer implementation to use based on the string *pathname* argument that it is provided. For example, an attempt to open or create a file which has an extension of `.aff` results in an AFF file being opened or created; opening a directory with a `.afd` extension results in the directory being treated as a collection of AFF files; calling `af_open()` with a path that has an an extension of `.afd` and the `O_CREAT` flag results in a directory being created. S3 files are specified with a URI in the form `s3://bucketname/prefix`. Split-raw files are automatically detected when a file is opened with a `.000` extension and a file is present with the same basename and a `.001` extension. EnCase files are specified with the standard `.E01` extension.

3.3   AFF Schema

The AFF schema defines specific segment names, their purposes, and the interpretation of the 32-bit flag and variable-length data areas. A list of the segments that have been defined as of AFFLIB v3.0.6 appears in Table 1. Because of the open nature of AFF, applications are able to create their own named segment and store that information in the AFF file.

Some of the more important AFF segments appear in Table 1.

Additional segment types used for integrity and privacy will be discussed later in this paper.

3.4   AFFLIB Streams Layer

In order to facilitate the integration of AFF into existing and new forensic software, AFFLIB implements a *streams layer* which provides a standard POSIX-like streams abstraction through a standard set of interfaces (Table 2).

Device Characteristics:

`pagesize`	The size of each AFF page, in bytes
`imagesize`	The number of bytes in the image
`sectorsize`	The size of each sector, in bytes
`devicesectors`	The number of sectors on the device.

Metadata:

`case_num`	Case number; for compatibility with EnCase.
`image_gid`	A randomly generated 128-bit number used to uniquely identify each acquired image.

Image characteristics:

`pagesize`	Size (in bytes) of each uncompressed AFF data page is stored in segment "flag" field.
`parity0`	The parity page; an XOR of all existing pages
`imaging_commandline`	The complete command used to create the image.
`imaging_date`	The date and time when the imaging was started.
`imaging_notes`	Notes made by the forensic examiner when the imaging was started.
`imaging_device`	The device that was used as the source of the image.
`blanksectors`	The number of sectors that are completely blank

AFF segments that are repeated for each page `%d`:

`page%d`	The named sector for each page of the disk image; `%d` is replaced with the page number, from 0 to $devicesectors \div \frac{pagesize}{sectors}$.
`page%d_md5`	The segment for the MD5 hash of the page
`page%d_sha1`	The segment for the SHA-1 hash of the page
`page%d_sha256`	The segment for the SHA256 hash of the page

Bad Sector Management:

`badsectors`	The number of sectors in the image which could not be read due to a hardware failure

Table 1: Some of the segment names used in the AFFLIB 3 schema.

AFFLIB POSIX-like functions

`af_open`	Opens an AFF/AFD/AFM/Encase/S3/raw/split raw file
`af_reopen`	Opens an existing file handle for reading or writing using the AFFLIB system
`af_popen`	Opens a process for reading or writing using the AFFLIB system
`af_read`	Read bytes from the file
`af_seek`	Seek to a different position in the disk image file
`af_tell`	Reports the current position in the disk image file
`af_eof`	Reports if the file pointer is at the end of the file
`af_write`	Write bytes to the file (used when imaging, not when performing forensic analysis)
`af_close`	Closes an AFF file

Nonstandard extensions:

`af_is_badsector`	Reports if the specified sector is bad
`af_set_error_reporter`	Establishes a callback function for alerting the operator that is called when AFF encounters an error
`af_set_cachesize`	Sets the size of the AFF page cache
`af_vstat`	Returns status information about the AFF implementation and the opened file
`af_stats`	Returns statistics about an AFF file
`af_set_option`	Sets an implementation option

Table 2: The AFF streams layer

3.5 Transparent Integration with AFUSE

Although support for AFF is relatively easy to add to an existing program by replacing calls to `open()`, `read()` and with `seek()` with `af_open()`, `af_read()`, and `af_seek()`, occasionally it is not possible or desired to make source code modification to forensic tools.

To accommodate these problems AFFLIB includes a user-level program called `affuse`. Implemented on top of the Filesystem in Usespace (FUSE) [Szeredi, 2008], `affuse` allows a compressed AFF file to appear as a raw file in the computer's own file system. FUSE takes care of automatically decompressing pages as necessary and caching the uncompressed pages with all available memory.

For example, if the user has an AFF file called `evidence.aff`, this can be made to appear as a raw file in the same file system with these commands:

```
# ls -l evidence.aff
-rw-r--r-- 1 simsong 555   409039930 Mar 23  2006 evidence.aff
# affuse evidence.aff evidenceraw
# ls -ld evidence*
-rw-r--r-- 1 simsong  555  409039930 Mar 23  2006 evidence.aff
drwxr-xr-x 2 root     root          0 Dec 31  1969 evidence.raw
# ls -l evidence.raw
total 0
-r--r--r-- 1 root root 2111864832 Dec 31  1969 evidence.aff.raw
```

Notice that the current FUSE implementation reports that the raw file occupies 0 blocks and has a time stamp of the Unix Epoc. A future version of `affuse` will make all of the named segments inside the AFF file visible in their own named files.

With `affuse`, any Linux forensics tool can access not only AFF files, but also EnCase files and files stored on S3. Windows can be run on the same workstation using VMWare Player [VMWare, 2008]. VMWare Player can be configured to allow the Windows operating system (and therefore Windows applications) to view the host computer's file system; with `affuse`, that file system can include the contents of an AFF evidence file.

4 AFFLIB 3 Integrity Features

AFFLIB 3 includes four important mechanisms for assuring the integrity.

1. Picewise hashing of image pages

2. Digital signatures of pages and all metadata

3. Parity pages

4. Chain-of-custody segments

The extensible design of the AFF storage system, allowed each of these features to be added to the original AFF specification [Garfinkel et al., 2006] without the need to make changes to the underlying AFF Data Storage Layer.

4.1 Piecewise hashing of data pages

AFF files store image data in special "page" segments which are typically 16MB in size. As each page segment is written, AFFLIB can automatically compute the page's MD5, SHA-1, and/or SHA256 hash and write an associated segment containing the hash value. The name of the hash page is simply the page name followed by the string `_md5`, `_sha1` or `_sha256`. Each hash may be individually enabled or disabled at runtime. For example, when SHA-1 piecewise hashing is enabled and the page `page3` is written, AFFLIB computes that page's SHA-1 and writes it into a segment named `page3_sha1`.

These piecewise hashes are used as a data integrity checks, similar to the way that the Expert Witness/EnCase format uses a CRC32. Even the MD5 is dramatically more secure than the CRC32. Nevertheless, these hashes are not intended to provide cryptographic protection for evidence: for that purpose AFF uses digital signatures, described below.

4.2 Digital Signatures

Digital signatures represent a significant improvement for evidence integrity over today's standard practice of recording the MD5 or SHA-1 of an imaged disk in an investigator's notebook:

- Unlike a hash code written into an investigator's report, digital signatures are mathematical structures created for the purpose of assuring the integrity of data: their suitability for this purpose have been considered for decades and is widely understood.

- By using standard digital signatures, it is possible to integrate digital electronic evidence with existing software that already understands how to process digital certificates.

- Unlike a hash code, which simply protects the image data, AFF digital signatures protect the entire disk image, and all of the associated metadata.

- The private key used to sign the signature can be tied to a specific device or investigator, allowing the signature to be used for non-reputability in addition to integrity.

- But the most important reason is that the use of digital signatures will permit the migration to imaging based on trusted hardware which can then help to assure the chain-of-custody of evidence from the system being imaged to the courtroom.

AFF computes digital signatures for both metadata and data. When computing signatures on metadata, the segment name, 32-bit argument, and metadata value is signed. In the case of digital signatures computed on image data ("pages"), the signatures are calculated on the uncompressed data. As a result, it is possible to acquire and digitally sign a disk image and then later compress the image without compromising the integrity of the digital signatures. Calculating the signature on the uncompressed data further assures that the compression algorithm does not modify the data between compression and decompression: if the data were modified, the signature would not validate.

AFFLIB uses OpenSSL to generate and verify all digital signatures; signatures are stored in PKCS#7 [Laboratories, 1993] format. Signatures that are stored directly in segments are stored as raw PKCS #7 objects, while signatures stored inside or adjacent to XML blocks are stored as Base64-encoded PKCS #7 objects.

AFF digital signatures complement the existing AFF integrity measures. Because the signature is stored in its own metadata segment, the signature does not change the content of the acquired disk image. And because AFM files can be used to annotate raw images, AFF signatures can be applied to raw image files without modifying the data itself. This is similar to PGP's ability to create "detached signatures," [Zimmermann, 1995] although it is more powerful because PGP's facility can only detect that an alteration has taken place, whereas AFF's signature facility can report which page has been modified.

Notice that AFF signatures are independent of the underlying storage system. The signatures can be stored in one file and the data in another file (as in an AFM file), or in multiple AFF files (as in an AFD directory). They can even be stored in the S3 network-based object storage system.

4.2.1 Signing AFF Segments

AFFLIB 3 allows each AFF segment to be individually signed. The signatures for these segments are stored in their own segments which are included as part of the AFF file.

The data in an AFF segment consists of three parts: the segment name, the 32-bit flag, and the segment bytestream. Because the name and the flag determine how the contents of the bytestream are interpreted, all three must be included in the computation of the signature.

AFFLIB 3 actually supports two signature modes, both of which include these three data elements. Both sign a hash of the segment data; the difference is how the hash is computed:

Signature Mode 0. The hash is computed from the segment name, a NULL byte, the segment argument (as a 32-bit number in network byte order), and the segment data.

Signature Mode 1. This mode is reserved for AFF data pages. The signature is computed by calculating the hash of the segment name, five NULL bytes, and the uncompressed page

data.[3] In this manner, the signature is computed over the original data, rather than data that has been compressed or otherwise processed.

As indicated above, the signatures are written into segments themselves, with the segment name being *name*/sha256 where *name* is the original segment name sha256 is the hash algorithm used for computing the signature. This format allows easy migration to signatures based on SHA512, should the need arise, or NIST's future signature algorithm. Indeed, the AFF signature format allows a single AFF file to be simultaneously signed with multiple schemes.

The observant reader will note that since AFF signatures are themselves stored in segments, it is possible that signatures themselves can be signed. While this is certainly a true observation, it is not a useful one, since the integrity of a signature is assured when the signature is validated.

4.2.2 Signing AFF files with X.509 certificates

Signatures can be written with either self-signed certificates or with X.509 [ITU, 2005] certificates that are issued as part of an organization's PKI. AFFLIB 3 uses the pluggable "EVP" signature support in the OpenSSL library [OpenSSL, 2008] to compute signatures; this library includes full support for both RSA and DSA X.509 certificates with 1024, 2048 or larger keys.

The easiest way to get a private key and a corresponding X.509 certificate is to make a self-signed certificate using the openssl command:

```
$ openssl req -x509 -newkey rsa:1024 -keyout sign.key -out sign.key -nodes
Generating a 1024 bit RSA private key
....................++++++
.......................++++++
writing new private key to 'sign.key'
-----
You are about to be asked to enter information that will be
incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or
a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:California
Locality Name (eg, city) []:Monterey
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Naval
Postgraduate School
Organizational Unit Name (eg, section) []:Department of Computer
```

---

[3]Five NULL bytes are used so that the data offset for the hash calculation is the same with Signature Mode 1 as it is for Signature Mode 0, which simplifies the implementation.

16

```
Science
Common Name (eg, YOUR name) []:Simson L. Garfinkel
Email Address []:slgarfin@nps.edu
$
```

When this command is run the user is asked a number of questions; OpenSSL uses the responses to these questions to build the CN field of the X.509 signing certificate and certificate request.

The contents of the certificate can be viewed with the `openssl x509 -text` command, as shown in Figure 2.

4.2.3   Certification of X.509 Certificates

As an alternative to creating a self-signed certificate, the practitioner can can create an RSA private/public key pair, create a certificate request (CSR), send the CSR to a certificate authority, and use the certificate that the authority returns. This procedure is the same procedure that the practitioner would use to obtain an X.509 key for email or running a secure web server [Housley and Polk, 2001, Adams and Lloyd, 2002].

4.2.4   Security for X.509 private keys

The `openssl` command presented in Section 4.2.2 places both the RSA private key, the public key, and the self-signed certificate into the same file. If the private key is stored without encryption, then the key file must be protected if non-repudiation is to be assured. Typically the contents of this file will be protected with the computer's operating system using the same mechanisms that are used to protect the computer's device drivers, operating system, and the afflib tools themselves: if these tools are secure, then so is the file containing the private key, and if these tools can be compromised, putting a passphrase on the private key adds little additional protection.

In some situations it is advantageous to have the private key stored separately from the operating system—for example, in a cryptographic device such as a smart card (*e.g.,* the Department of Defense Common Access Card [US Department of Defense, 2008]) or a USB token. Although OpenSSL has support for these devices, we have not implemented this functionality at the AF-FLIB level due to our limited development resources. A future version of AFFLIB can support this functionality if it is required by AFF users.

4.3   Bill of Materials and Chain-of-Custody

AFFLIB 3 introduces a special XML structure that contains a list of every AFF segment in the file, a signature for each segment, a set of "notes," and a public key. This structure is called an "AFF Bill Of Materials" (AFFBOM). An example of the XML structure appears in Figure 3.

When an AFF image is created with `aimage`, `afconvert`, copied with `afcopy`, or signed with `afsign`, an AFFBOM is created and signed with the private key belonging to the person

```
$ openssl x509 -text -in sign.key -noout

Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            a3:e1:ef:44:63:04:74:00
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=US, ST=California, L=Monterey, O=Naval Postgraduate School,
            OU=Department of Computer Science,
            CN=Simson L. Garfinkel/emailAddress=slgarfin@nps.edu
        Validity
            Not Before: May 17 01:40:13 2008 GMT
            Not After : Jun 16 01:40:13 2008 GMT
        Subject: C=US, ST=California, L=Monterey, O=Naval Postgraduate School,
            OU=Department of Computer Science,
            CN=Simson L. Garfinkel/emailAddress=slgarfin@nps.edu
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:be:4e:10:cc:e4:ae:76:c2:d1:7c:72:c7:74:32:
                    f3:43:04:51:ed:ba:ed:a4:26:4d:46:b8:98:6c:bc:
                    28:10:13:7c:7d:20:a7:69:c7:9d:f1:66:4c:d3:b1:
                    12:48:fc:07:2d:87:83:f3:e4:0c:c8:64:b2:38:6a:
                    4a:18:39:bf:3f:08:ba:37:e1:69:3f:57:0c:06:8a:
                    c6:95:9d:f5:4a:62:fd:4d:04:49:f1:f7:23:b0:e3:
                    e4:ad:41:a1:4a:64:78:d2:fb:16:3d:22:2f:e1:59:
                    0d:47:07:85:1a:e7:aa:fa:3b:61:fe:0f:56:21:48:
                    c3:e1:49:c5:ad:32:08:4d:57
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                AE:A6:63:40:52:BF:08:1D:E1:D3:A5:85:75:16:D8:BD:76:71:1E:BB
            X509v3 Authority Key Identifier:
                keyid:AE:A6:63:40:52:BF:08:1D:E1:D3:A5:85:75:16:D8:BD:76:71:1E:BB
                DirName:/C=US/ST=California/L=Monterey/O=Naval
             Postgraduate School/OU=Department of Computer Science/CN=Simson
             L. Garfinkel/emailAddress=slgarfin@nps.edu
                serial:A3:E1:EF:44:63:04:74:00

            X509v3 Basic Constraints:
                CA:TRUE
    Signature Algorithm: sha1WithRSAEncryption
        34:6d:22:50:28:72:3b:e5:4d:fd:99:3f:79:6a:37:e0:75:45:
        fb:df:a5:c8:29:a5:4d:62:3f:58:8a:a6:1a:48:86:83:c7:03:
        d7:59:84:b9:e5:67:2b:2b:7a:8a:13:72:ec:82:d0:9a:56:b3:
        fd:a5:8a:7f:c1:68:6a:db:ea:d2:1f:41:b9:ab:23:16:f1:59:
        ca:91:3d:cb:fc:58:08:01:ab:4b:7b:15:c5:c5:7a:fc:a9:e8:
        ea:09:fc:8d:4f:1b:68:a7:e5:34:19:9d:ed:73:46:e5:95:87:
        3e:e2:65:58:0f:a2:66:d3:a5:6f:62:47:78:e8:65:34:30:b4:
        49:9d
```

Figure 2: The OpenSSL command can be used to decode the contents of a certificate.

who did the acquisition. This is stored in a special segment called `affbom0`.

Of course an individual AFFBOM segment can be removed from an AFF file; indeed, all of the signatures can be removed as well. This is not a shortcoming specific to the AFF signature scheme: any digital signature scheme suffers from the shortcoming that signatures can be stripped and the content can be changed by an adversary. Signatures can be stripped from signed code, producing code that is unsigned. Signatures can be removed from S/MIME signed email messages, producing conventional, unsigned email message. There is, in principle, no way to tell the difference between an object that has had its signature removed from one that was never signed in the first place. The only way to know that a signature has been removed is through the use of policy—for example, a protocol that prohibits an organization from releasing an unsigned data object. But even then, there is no way to tell the difference between a genuine data object that was released and later had its signature stripped, and a fraudulent data object that was never signed in the first place.

4.3.1 AFF Bill of Materials

When an AFF file is created or copied, an AFF Bill of Materials can be added. This block is an XML data structure that includes the date that it was signed, the certificate used to create the signature, notes, and an array of elements that represent each segment in the AFF file. An example of the schema appears in Figure 3

The segment is called a *bill of materials* because it is literally a parts list of all the segments that make up the specific AFF file. Since AFF files are segmented and segments can be added or removed at will, the need exists for a single structure that lists all of the segments that need to be present for a file to be complete. Without the AFFBOM segments could be added, removed, or changed without detection.

The AFFBOM contains XML elements for the date that the signature was created, the program that created the signature, human-readable notes, and an XML array containing a cryptographic hash of each AFF segment in the AFF file. Hashes can be computed in mode 0 or mode 1, as discussed in Section 4.2.1. At the end of the XML structure is stored a Base64-encoded digital signature of the structure.

What makes it possible to detect change is not just the fact that there is an AFFBOM, but the fact that it is digitally signed. Provided that the private key is guarded and that the signature process is trusted, a relying party can be assured that the specific set of segments with the specific hashes existed in the AFF file at the time the signature was created.

The XML block is signed using the OpenSSL signature routines; the resulting signature is placed at the end of the XML block as a base-64 coded PKCS #7 object. Although the W3C XML-Signature Recommendation [Bartel et al., 2002] might have been a better choice, we were unable to find a suitable implementation, and the complexity of the specification is such that we did not wish to attempt writing it ourself. (The W3C reference implementation contains more

19

```
<affbom version="1">
  <!--Date XML was written:-->
  <date type="ISO 8601">19980708T13:33:11</date>
  <!--Base64 encoding of certificate used to sign CCB-->
  <program>afcopy</program>
  <signingcertificate>
    YXNkYXNkZgphZHN...
    ...s39fjasl3JSFCmYK
  </signingcertificate>
  <notes>
      Human-readable notes from the examiner
  </notes>
  <!--What follows is an array of elements, one for each AFF segment.-->
  <affsegments>
    <segmenthash segname='myname1' mode='0' alg='sha256'>
      base64 encoding of the hash of the named segment
    </segmenthash>
    ...<!--multiple segmenthash segments will be present-->
  </affsegments>
</affbom>
7zzW9WJ07RPuTH4G291b6YSW5SUQacD7UGJTiwpA+NgPm6/RRoJwSQcud6RxwkkL
thQrN0poqv8T8U7p8cSiuphrL29oBY9J4okjv1xXTdLoHoaf5Ft6kt+QqeSX4bOB
...
```

Figure 3: The AFF Bill of Materials (AFFBOM) with signature at end.

than 130 C source files and requires two additional open source packages for proper operation.)
Our implementation has the advantage of being small, easy to validate, and implemented. It
would not be difficult to migrate to XML-Signature if such functionality is required, however.

4.3.2 Providing a Chain of Custody

Each time a signed AFF file is copied with `afcopy`, a new AFFBOM can be created which
includes a new AFFBOM that covers all of the original segments and all of the previous AFF-
BOMs. In this manner the sequence of signed bill-of-materials becomes a custody chain, show-
ing who has copied the image and verifying that no evidentuary segments have been added,
deleted, or modified. These AFFBOMs are stored in segments named `affbom1`, `affbom2`,
etc., where the number is incremented for each copy generation.

The AFFLIB source code contains a demonstration script called `test_signing.sh` that
creates an evidence file and three X.509 certificates: one for Mr. Agent, one for Ms. Analyst,
and one for Dr. Librarian, all officials in the fictional town of Remote, CA. The evidence file
`rawevidence.iso` is converted into a file `evidence.aff` with `afconvert` and then
signed with `afsign` using this command:

```
$ afsign -s agent.pem evidence.aff
```

The signature can be verified using the `afverify` command:

```
$ afverify evidence.aff
```

Notice that the `afverify` command does not need the user to specify a certificate to use for verification, because the signing certificate is embedded in the `evidence.aff` file. When the program runs it displays the certificate that was used for verification, so that the investigator can verify that the file is still signed with the correct certificate.

Mr. Agent transfers the evidence to Ms. Analyst. This is done with the `afcopy` command:

```
$ afcopy -n -s analyst.pem evidence.aff evidence2.aff
Enter notes. Terminate input with a '.' on a line by itself:
This copy was made by the analyst.
.
Thank you.
Copying evidence.aff --> evidence2.aff
evidence2.aff: 20017252 bytes transfered in 10.07 seconds. xfer rate: 1.99 MByte
```

Notice that the file is automatically signed because a public/private keypair is provided in the file `analyst.pem`. The `-n` option tells `afcopy` to take a note from standard input.

Of course, an AFF file can still be copied without using the `afcopy` command. In this case the file will be copied without a new XMLBOM segment being added.

4.3.3 Verification

Verification is done with the AFFLIB program `afverify`. This program opens the requested AFF file and scans for the `affbom`*n* segments. For each segment the program then verifies the signature on the XML block, then opens each AFF segment and verifies that segment's cryptographic hash. (Segment hashes are cached after they are computed for efficiency.)

The `afverify` program can report:

- Missing segments that were signed but are now missing.

- Extra segments that were not signed but have been added to the file.

- AFF segments whose signature no longer verify.

- AFF segments that were modified at one point during the conveyance of evidence. These will appear as segments that do not verify for older AFFBOMs but to verify for later AFFBOMs. In this manner it is possible to determine when the segment was modified.

Figure 4 shows `afverify` applied to the file `evidence2.aff` created in the previous section.

```
Filename: evidence2.aff
# Segments signed and Verified:      11
# Segments unsigned:                  0
# Segments with corrupted signatures: 0

SIGNING CERTIFICATE :
   Subject: C=US, ST=California, L=Remote, O=Country Govt., OU=Sherif Dept,
        CN=Mr. Agent, emailAddress=agent@investiations.com
   Issuer: C=US, ST=California, L=Remote, O=Country Govt., OU=Sherif Dept,
        CN=Mr. Agent, emailAddress=agent@investiations.com


Number of custody chains: 2
---------------------
Signed Bill of Material #1:

SIGNING CERTIFICATE :
   Subject: C=US, ST=California, L=Remote, O=Country Govt., OU=Sherif Dept,
        CN=Mr. Agent, emailAddress=agent@investiations.com
   Issuer: C=US, ST=California, L=Remote, O=Country Govt., OU=Sherif Dept,
        CN=Mr. Agent, emailAddress=agent@investiations.com

Date: 2008-04-26T11:06:06
Notes:

---------------------
Signed Bill of Material #2:

SIGNING CERTIFICATE :
   Subject: C=US, ST=California, L=Remote, O=State Police, OU=Forensics,
        CN=Ms. Analyst, emailAddress=analyst@investiations.com
   Issuer: C=US, ST=California, L=Remote, O=State Police, OU=Forensics,
        CN=Ms. Analyst, emailAddress=analyst@investiations.com

Date: 2008-04-26T11:06:21
Notes:
This copy was made by the analyst.

---------------------

EVIDENCE FILE VERIFIES.
```

Figure 4: `afverify` applied to file `evidence2.aff` created as part of the AFFLIB test routines.

4.4 AFF Parity Pages

To allow for the recovery of data after corruption or data loss, AFFLIB 3 introduces the concept of *AFF Parity Pages*. Similar to parity drives used in a hard drive storage array, the AFF parity page is written for each disk image file at the conclusion of disk imaging; each byte of the parity page is computed by taking the XOR of the corresponding byte of all the other disk pages in the AFF file. Thus, the contents of any other page can be reconstructed simply by taking the XOR of all the remaining pages and the parity page.

AFF parity pages work with piecewise hashes and digital signatures to provide enhanced data recovery. If the hash or digital signature indicates that a page has been corrupted, that page can be erased and then reconstructed using all of the other AFF pages and the parity page. Once reconstruction is complete, the signature or page hash (which are stored in a different location) can be used to determine if the reconstruction is correct.

Parity pages are automatically created when an image is signed with the `afsign` utility. They can also be created by the `aimage` disk imaging utility, which was previously part of AFFLIB but is now its own standalone distribution.

Because they are the same size as the data pages, parity pages are not limited to correcting a single error. Indeed, the combination of parity pages and per-page hashes and/or signatures allows a wide number data corruption events to be not only detected but corrected, including:

- One or more bytes changed within a single page.

- One of more bytes changed across multiple pages, provided that bytes with the same offset are not modified on different pages.

Finally, overlapping ranges of bytes on multiple pages that are damaged can be reconstructed using a brute force operation. In these cases multiple "trail reconstructions" must be attempted, with each reconstructed tested by computing the pages' hash and seeing if the hash matches the hash that was previously calculated. Essentially, this approach uses a brute force search for the correct data: once the correct set of bytes is found, the signatures validates. In practice such an approach would only work if the overlap region in each page is confined to 4 bytes or less; beyond that, the computational overhead is simply too great. If entire sectors are corrupt or missing, reconstruction will not be successful. (Such a reconstruction is not currently implemented by the `afverify` command, but may be in a future version.)

4.5 Signed raw files

AFF's AFM format allows a disk image to be stored in an uncompressed raw file (eg `file.iso`) and the associated metadata to be stored in a `.afm` file. The AFM format can also handle raw data stored as a series of split raw files (eg `file.001`, `file.002`, *etc.*).

Beacuse AFF tools operate on named segments that are independent of the underlying storage container, the AFM format allows any ISO-file to be signed using the `afsign` command. The `afsign` program will automatically detect if it is signing a raw file and will create a `.afm` file to hold the signature. When `filename.iso` is signed, the `afsign` create a new file called `filename.afm` which contains the signatures, the signed bill of materials, and other metadata:

```
$ ls -l myfile.iso
-rw-r--r--  1 simsong  simsong  63107908 Apr 26 11:30 myfile.iso
$ ./afsign -s agent.pem myfile.iso
Signing segments...
Calculating BOM for page0...
Calculating BOM for page1...
Calculating BOM for page2...
Calculating BOM for page3...
$ ls -l myfile*
-rw-------  1 simsong  simsong  16785481 Apr 26 11:30 myfile.afm
-rw-r--r--  1 simsong  simsong  63107908 Apr 26 11:30 myfile.iso
$
```

Although it is also possible to sign ISO files using existing tools such as PGP with detached signatures, `afsign` has several advantages:

- `afsign` will sign every 16 megabytes chunk of the ISO file. In this way, if the file is corrupted, it is possible to pinpoint what data is invalid and what data is still good.

- Unlike PGP, `afsign` allows the addition of notes and other metadata when a signature is written.

- `afsign` utilizes X.509 certificates, allowing easy integration into existing PKI-based systems.

- Because `afsign` also computes a parity page, it is possible to repair a damaged raw file using `afrecover` (as discussed in Section 4.4).

Figure 5 illustrates recovery of a corrupted file. First the file is corrupted with a block of random data. Next the file is checked with `afverify`. Finally the file is recovered using `afrecover` (Figure 5).

25

```
$ dd if=/dev/random of=myfile.iso count=1 skip=1 conv=notrunc
$ afverify myfile.afm
Filename: myfile.afm
# Segments signed and Verified:       13
# Segments unsigned:                   0
# Segments with corrupted signatures: 1

SIGNING CERTIFICATE :
   Subject: C=US, ST=California, L=Remote, O=Country Govt., OU=Sherif Dept,
            CN=Mr. Agent, emailAddress=agent@investiations.com
   Issuer: C=US, ST=California, L=Remote, O=Country Govt., OU=Sherif Dept,
            CN=Mr. Agent, emailAddress=agent@investiations.com

Bad signature segments:
page0

Number of custody chains: 1
--------------------
Signed Bill of Material #1:

SIGNING CERTIFICATE :
   Subject: C=US, ST=California, L=Remote, O=Country Govt., OU=Sherif Dept,
            CN=Mr. Agent, emailAddress=agent@investiations.com
   Issuer: C=US, ST=California, L=Remote, O=Country Govt., OU=Sherif Dept,
            CN=Mr. Agent, emailAddress=agent@investiations.com

Date: 2008-04-26T11:35:18
Notes:

--------------------

EVIDENCE FILE DOES NOT VERIFY; EVIDENTUARY VALUE MAY BE COMPROMISED.
$ afrecover myfile.afm
myfile.afm has a bad signature
Attempting to repair page0
Page page0 successfully repaired
$
```

Figure 5: Demonstration of file corruption and recovery using `afrecover`.

5 AFFLIB 3 Encryption Features

AFFLIB 3 introduces the ability to encrypt AFF evidence files with the AES-256 algorithm [NIST, 2001].

Each segment of each AFF file may be optionally encrypted with a unique, randomly generated 256 bit AES *session key*. This key can then itself be encrypted using a passphrase provided by the user or encrypted with an X.509 public key. Because of this two-step process, the passphrase or public key used to encrypt an AFF file can be changed in just a few seconds without having to decrypt and re-encrypt the entire disk image.

Whereas some other forensic programs provide the ability to put a "password" on an evidence file, those passwords can be disregarded by non-conformant programs. (For example, Get-Data claims that it's MountImage Pro program can "open EnCase password protected image files without the password." [GetData Software, 2008] `Libewf` allows the user to ignore the passphrase for EnCase images when the images are opened.) AFFLIB 3 uses true encryption: if you do not know the correct decryption key, the only way to access the evidence is to brute-force the encryption passphrase or the X.509 private key.

5.1 AFF Encryption Schema

Similar to the design of AFF Signatures, AFF Encryption is layered on top of the basic AFF functionality that stores name/value pairs.

Three encryption layers are implemented:

1. **AFF Base Encryption**, which provides encryption of the AFF segment contents, but not the segment names or the segment flags. All of the segments in an AFF file are encrypted with a the same randomly generated *affkey*.

2. **AFF Passphrase Encryption**, a scheme for storing the AFF file's affkey in an AFF segment that is itself encrypted with a passphrase.

3. **AFF Public Key Encryption**, which stores the AFF file's *affkey* in an AFF segment that is encrypted with an X.509 public key.

5.1.1 AFF Base Encryption

Encrypted AFF segments are stored in segments where name is generated by taking the name of the unencrypted segment and appending a slash followed by the encryption algorithm and keysize. For example, whereas the first 16MB of a disk image are typically stored in a segment named `page0`, in an encrypted AFF file the page is named `page0/aes256`.

As discussed above, a single AFF session key is used to encrypt all of the AFF segments in a given file. In AFFLIB 3 this key is randomly generated and is not accessible to the user.

Encryption is implemented as modifications to the `af_update_seg` and `af_get_seg` functions inside the `lib/afflib.cpp` source file:

- When a program linked with AFFLIB attempts to store a segment, AFFLIB checks to see if an encryption key has been set; if one has, the segment's content is encrypted and the segment is stored at the modified name (*e.g.,* `page0/aes256` instead of `page0`).

- When a program linked with AFFLIB attempts to fetch a segment and the segment does not exist, AFFLIB checks to see if an encryption key is set. If one is, AFFLIB attempts to fetch the segment with the modified name. If the segment can be fetched, AFFLIB attempts to decrypt the segment with the key that has been set. If decryption is successful the data is returned to the caller.

By implementing encryption at this layer, we provide for data to be encrypted after it is compressed by the AFFLIB page system. This is the preferred approach, as data cannot be compressed after it is encrypted.

If a key is set, then pages that are written are automatically encrypted, then written to the data store.

If an unencrypted page is updated and encryption is enabled, the encrypted page is first written, then the unencrypted page is deleted. The delete operation involves overwriting the unencrypted segment with NULLs inside the AFF file. Multiple overwrites are not implemented, as they are not required to preserve data privacy on modern hardware [NIST, 2006].

It is an error to change the affkey encryption key once it has been set.

5.1.2 Encryption modes and blocking

Encryption is performed with Cipher Block Chaining mode. The initialization vector is the name of the sector padded with NULLs. Every segment in an AFF file has a different segment name, thus a different IV. (IVs do not need to be kept secret to ensure privacy; the sole purpose of the IV is to assure that different pages with the same data nevertheless have different encryptions.)

Block ciphers such as AES require that all buffers be padded to the block size; with AES the block size is 16 bytes. For performance AFF does not add padding if the page is already a multiple of the block size. If the size of the vector is not a multiple of the AES block size, two values are computed:

$$\textbf{extra} \;=\; \textbf{len}(\bmod\ \textbf{blocksize}) \tag{1}$$
$$\textbf{pad} \;=\; 16 - \textbf{extra} \tag{2}$$

The buffer is padded by **pad** bytes; the buffer is now a multiple of the AES block size. The buffer is encrypted. Finally, **extra** pad bytes are appended. Although the buffer is expanded, it is now possible to recover the original length of the buffer when the segment is read and decrypted.

To decrypt the buffer and recover the original length, the values **extra** and **pad** are computed once again. The **extra** pad bytes are removed, the buffer is decrypted, and last **pad** bytes are removed. The length of the resulting buffer is set to be the length of the encrypted buffer minus the AES block size, and the decrypted data buffer is returned. In this way, the length does not need to be explicitly coded. This scheme is the same as the one employed by PKCS #7 ([Laboratories, 1993]; in keeping with PKCS #7, the pad byte is hex `01` if one pad byte is required, hex `02 02` if two bytes are required, and so on.

The integrity of decrypted page data can be checked by comparing the MD5 of the decrypted `page`*n*`/aes256` segment with the decrypted contents of the `page`*n*`_md5/aes256` segment using the `afverify` command, or by verifying the AFF signatures if they are present.

5.1.3   Design Limitations

There are a number of limitations that arise from the way that AFFLIB 3 implements encryption. In this section we will briefly discuss the limitations and explain why we think they are inconsequential.

- **AFF Encryption only encrypts the bytestream of segments; the segment name and 32-bit flag are unencrypted.**

  AFF encryption is created for the specific purpose of encrypting data and metadata that are acquired from disk images. For this reason, we concluded that there was no reason to attempt to obscure the segment names or 32-bit flags with cryptography, because these do not hold information that needs to be kept confidential.

- **A single AFF file may contain information that is both encrypted and not encrypted.**

  Because encryption is performed on a per-segment basis, it is possible to have segments that are both encrypted and unencrypted. We see this ability as an advantage, as it allows files that are unencrypted to be encrypted in place without the need to allocate double the disk space. Should the encryption process be interrupted (for example, by a power failure), the process can continue where it left off at some later point.

- **The *affkey* cannot be changed once it is set for a specific file.**

  We believe that the added complexity to support multiple *affkeys* within a single file would not be worth the complexity. In part this is because the key is not intended to be used by the user: it is really just a session key that is used by the passphrase or the public key encryption system. Should the *affkey* be compromised, every segment would need to be

29

reencrypted. The easiest way to do this would be to copy one encrypted AFF file to another file using the `afcopy` command.

5.2 AFF Passphrase Encryption

Most investigators would prefer to work with a simple passphrase than with a 256-bit encryption key that needs to be specially maintained, so AFFLIB 3 provides this ability as well.

AFF Passphrase Encryption builds upon the Base Encryption. When a passphrase is entered AFFLIB uses the SHA256 algorithm to change the passphrase into a 256-bit hash. But instead of using this has as an encryption key directly, the hash is used to encrypt the randomly generated *affkey*. The encrypted session key is then stored in the `affkey-aes256` segment.

This scheme could easily be modified to support multiple passphrases on each file, storing them in segments such as `affkey-aes256_0`, `affkey-aes256_1`, *etc.*, although there have been no requests for such functionality.

The contents of the `affkey_aes256` segment is a 68 byte structure:

bytes	purpose
0–3	The version number, stored in network byte order.
4–67	The affkey, encrypted with AES in Electronic Codebook (ECB) mode using SHA-256 of the passphrase as the encryption key.
68–131	The SHA-256 of the affkey (for verification purposes).

With this scheme the passphrase can be changed without requiring the entire disk image to be re-encrypted—all that needs to be done is that the `affkey-aes256` segment is read, decrypted using the old passphrase, and and re-encrypted with the new passphrase. (If a disk image does need to be re-encryped—for example, if the *affkey* is compromised—this can be easily done by copying the file with the `afcopy` command from one AFF file to another.)

A further advantage of this scheme is that the passphrase is not cached in memory.

5.3 AFF Public Key Encryption

AFF's public key encryption facilities allow a disk image to be encrypted when it is created with a public key; to use the disk image at a later time requires the corresponding private key. This might be useful if an image is to be acquired in the field: once the image is acquired, it would be cryptographically protected so that it could not be deciphered even if the machine (or person) doing the encryption was later intercepted.

In practice, the disk image's public key is specified when the file is created. At this point a random *affkey* is created, encrypted with the public key, and cached in memory. As long as the file remains open it can be read and written. But when the file is closed the in-memory copy of the *affkey* is erased. Thus, once the file is closed, access to the data for either reading or writing requires the corresponding private key.

Public key encryption is implemented by taking the affkey and encrypting it using the OpenSSL "envelope" provisions. This involves creating a random session key and initialization vector, encrypting the affkey with the session key using a block cipher, and then encrypting the session key with the public key that will be used for sealing. The resulting encrypted session key, encrypted affkey, and initialization vector are all stored it in a segment called `affkey-evp`*n* where *n* starts at 0 and increases. Padding is according to PKCS #1 [Laboratories, 2002]. In this manner the same affkey is never encrypted twice with two different RSA public keys.

For encrypting, the public key used for sealing can be specified in one of two ways:

- In a file whose name is provided on the command line, using the "-C" option ("C" for Certificate).

- The filename referenced by the environment variable `AFFLIB_ENCRYPTING_PUBLIC_KEYFILE`

For decrypting, the private key used for unsealing can be specified in one of two ways:

- In a file whose name is provided on the command line, using the "-K" option ("K" for Key).

- The environment variable `AFFLIB_DECRYPTING_PRIVATE_KEY`.

Although AFFLIB does not currently support the entering of a passphrase to decrypt private keys that protected with a passphrase or for using a smart card or cryptographic token, these capabilities can be added to a future release if requested by users. OpenSSL already has support for these capabilities; all that is required is passing this capability through to the AFFLIB API).

5.4 Integrating encryption with existing tools

5.4.1 Specifying a passphrase as part of a filename

AFFLIB understands Uniform Resource Identifier [Berners-Lee et al., 2005] (URI) syntax, and URIs have provisions for specifying passwords. Thus, it is relatively straightforward to integrate passphrase-protected AFF files with existing command-line forensic tools by simply specifying the passphrase as part of the filename.

URIs such as `http://www.afflib.org/download/afflib.tar.gz` consists of *scheme* (*e.g.,* `http`), an *authority* (`www.afflib.org`), a path (`download/afflib.tar.gz`), a

query and a fragment (not shown here). Although the *authority* is typically just a hostname, the full syntax for the authority is:

```
authority = [ userinfo "@" ] host [ ":" port]
```

Userinfo was traditionally represented as `username:password`. Although this syntax is deprecated in the current version of RFC3986, to avoid for the possible leakage of confidential information, we have chosen to use it to provide forensic workers with an easy means of specifying passwords on the command line.

A file can be encrypted using AFF `afcopy` command like this:

```
$ afcopy myfile.iso file://:password@/myfile.aff
```

The resulting file can only be accessed if the passphrase is used:

```
$ afcat myfile.aff\verb|wc
afcat: This file has 5 encrypted segments.
afcat: No unencrypted pages could be found.
        0         0         0
$ afcat file://:password@/myfile.aff | wc
 5481881 5980668 63107908
```

5.4.2   Specifying a passphrase in an environment variable

As an alternative to specifying the passphrase on the command line, AFFLIB 3 allows passphrases to be specified in the `AFFLIB_PASSPHRASE` environment variable:

```
$ export AFFLIB_PASSPHRASE=password
$ afcat myfile.aff | wc
 5481881 5980668 63107908
$
```

5.4.3   Using encryption with **affuse**

Finally, an encrypted image can be mounted using `affuse`; the decryption is done in the user-level `affuse` program, so that operating system (and application program) are able to directly process unencrypted, uncompressed data:

```
# affuse file://:password@/myfile.aff mnt
# ls -l mnt
total 0
-r--r--r-- 1 root root 67108864 Dec 31  1969 myfile.aff.raw
```

Notice that this command must be executed as root. Also note that modification time of the raw file is incorrectly set to the Unix epoch in the current implementation.

6 Signature and Encryption API

This section describes the API for the AFF encryption layer.

AFF Base Encryption

`af_set_aes_key`	Sets the *affkey* that will be used for the currently open AFF file. Returns an error if the key is already set.
`af_cannot_decrypt`	Returns true if there are encrypted pages present that cannot be decrypted with the currently specified *affkey*.
`af_has_encrypted_segments`	Returns true if the currently open AFF file has encrypted segments.
`af_is_encrypted_segment`	Returns true if *segname* is encrypted.

AFF Passphrase Encryption

`af_establish_aes_passphrase`	If no key has been set, creates a random *affkey*, encrypts the key with the *passphrase* and stores the segment in the AFF file. Returns an error if a key has already been set.
`af_change_aes_passphrase`	Changes the passphrase for an AFF file from *oldphrase* to *newphrase*. Returns an error if *oldphrase* is not the correct phrase.
`af_use_aes_passphrase`	Tests to see if *passphrase* is in fact the correct passphrase for the currently opened AFF file. If it is, the passphrase will be used. An error is returned otherwise.

AFF Public Key Signatures

`af_set_sign_files`	Opens the files containing a private key and certificate. The cryptographic information they contain are thereafter used to sign all segments that are updated.
`af_sign_seg`	Asks AFF to sign a specified segment.
`af_sign_all_unsigned_segments`	Asks AFF to sign all of the unsigned segments.
`af_is_signed_segment`	Returns TRUE if there is a signature segment for the segment *segname*.

AFF Public Key Encryption(Sealing)

`af_set_seal_certificates`	Creates an *affkey*, encrypts the key with each of the provided X.509 certificates, and stores each encrypted *affkey* in its own segment
`af_set_unseal_keybuffer`	Specifies a string buffer containing an unencrypted RSA key in PEM format.

7 Conclusion

This paper introduces the provisions for cryptographic security, integrity, and chain-of-custody that have been incorporated in Version 3 of the Advance Forensic Format Library (AFFLIB). These provisions build upon the AFF format introduced by Garfinkel *et. al* in 2006 [Garfinkel et al., 2006], allowing transparent access to evidence files that are digital signed or encrypted.

Compared with other approaches and alternatives, AFF Signatures and Encryption offers the following advantages:

- The scheme was simple to implement and test.

- AFFLIB offers real encryption of evidentiary data, not a simple "password" as is present in other systems.

- Raw files can be signed without the need to modify the original data.

- Unencrypted evidence files can be encrypted in-place.

Because of design decisions, AFFLIB encryption does have a few disadvantages. Specifically:

- Segment names and the 32-bit argument stored with AFF segments are digitally signed but they are not encrypted. Since segment names and the 32-bit argument never hold evidentiary data or metadata, this lack of encryption is not considered to be significant.

- Each AFF file is encrypted with its own key; the only way to change the key is to copy the data from one encrypted file to another. However, the passphrase used to encrypt a file can be changed instantly.

- AFFLIB caches the encryption key in memory in the AF structure, allowing the key to be stolen by hostile software. This shortcoming can overcome through the use of trusted operating systems or cryptographic tokens.

7.1 Future Work

We continue to make improvements in AFF and `aimage`. More information about AFF, including the source code for AFFLIB 3, can be found at `http://www.afflib.org/`.

35

7.2 Acknowledgments

AFF and AFFLIB are trademarks of Simson L. Garfinkel and Basis Technology, Inc.

References

[Adams et al., 2001] Adams, C., Cain, P., Pinkas, D., and Zuccherato, R. (2001). Internet x.509 public key infrastructure time-stamp protocol (tsp).

[Adams and Lloyd, 2002] Adams, C. and Lloyd, S. (2002). *Understanding PKI: Concepts, Standards, and Deployment Considerations*. Addison-Wesley Professional, 2 edition.

[Amazon, 2008] Amazon (2008). Amazon simple storage service (amazon s3). Amazon Web services. `http://aws.amazon.com/s3`.

[ASR, 2002] ASR (2002). Expert witness compression format specication. ASR Data Acquisition and Analysis. `http://www.asrdata.com/SMART/whitepaper.html`.

[Bartel et al., 2002] Bartel, M., Boyer, J., Fox, B., LaMaccia, B., and Simon, E. (2002). Xml-signature syntax and processing. W3C. `http://www.w3.org/TR/xmldsig-core/`.

[Berners-Lee et al., 2005] Berners-Lee, T., Fielding, R., and Masinter, L. (2005). RFC 3986: Uniform resource identifier (uri): Generic syntax.

[Boneh, 2001] Boneh, D. (2001). Cryptographic hashing. `http://crypto.stanford.edu/~dabo/courses/cs255_winter01/1-hashing.pdf`, Course notes for CS255 Winter 01.

[Carrier, 2006] Carrier, B. (2006). *A Hypothesis-Based Approach to Digital Forensic Investigations*. PhD thesis, Purdue University.

[Computer Systems Laboratory and Technology, 1993] Computer Systems Laboratory, N. I. o. S. and Technology (1993). FIPS-180 secure hash standard. U.S. Department Of Commerce. Also known as: 58 Fed Reg 27712 (1993).

[Dierks and Allen, 1999] Dierks, T. and Allen, C. (1999). RFC 2246: The TLS protocol version 1. Status: PROPOSED STANDARD.

[Diffie and Hellman, 1976] Diffie, W. and Hellman, M. E. (1976). New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654. `citeseer.ist.psu.edu/diffie76new.html`.

[Fonseca, 2007] Fonseca, B. (2007). Hard-drive changes: Long block data standard gets green light. *Computerworld*. `http://www.computerworld.com/action/article.do?command=printArticleBasic&articleId=9018507`.

[Friedl, 2005] Friedl, S. (2005). An illustrated guide to cryptographic hashes. `http://www.unixwiz.net/techtips/iguide-crypto-hashes.html`.

[Garfinkel, 2008] Garfinkel, S. L. (2008). Afflib. `http://www.afflib.org/`.

[Garfinkel et al., 2006] Garfinkel, S. L., Malan, D. J., Dubec, K.-A., Stevens, C. C., and Pham, C. (2006). Disk imaging with the advanced forensic format, library and tools. In *Research Advances in Digital Forensics (Second Annual IFIP WG 11.9 International Conference on Digital Forensics)*. Springer.

[GetData Software, 2008] GetData Software (2008). GetData Software Developoment Company. `http://www.mountimage.com/`.

[Haggerty and Taylor, 2007] Haggerty, J. and Taylor, M. (2007). *FORSIGS: Forensic Signature Analysis of the Hard Drive for Multimedia File Fingerprints*, pages 1–12. Springer. `http://www.springerlink.com/content/21478kr877478805/`.

[Harbour, 2006] Harbour, N. (2006). dcfldd. `http://dcfldd.sf.net`.

[Housley and Polk, 2001] Housley, R. and Polk, T. (2001). *Planning for PKI: Best Practices Guide for Deploying Public Key Infrastructure*. Wiley.

[ICS, 2008] ICS (2008). Secure hash signature generator. Intelligent Computer Solutions. `http://www.ics-iq.com`.

[ITU, 2005] ITU (2005). Recommendation x.509 (08/05): The directory: Public-key and attribute certificate frameworks. International Telecommunication Union. `http://www.itu.int/rec/T-REC-X.509-200508-I`.

[Keightley, 2003] Keightley, R. (2003). EnCase version 3.0 manual revision 3.18. Guidance Software. `http://www.guidancesoftware.com/`.

[Kloet et al., 2008] Kloet, B., Metz, J., Mora, R.-J., Loveall, D., and Schreiber, D. (2008). libewf: Project info. Uitwisselplatform.NL. `http://www.uitwisselplatform.nl/projects/libewf/`.

[Kohnfelder, 1978] Kohnfelder, L. M. (1978). Towards a practical public-key cryptosystem. Undergraduate thesis supervised by L. Adleman.

[Laboratories, 1993] Laboratories, R. (1993). Pkcs #7: Cryptographic message syntax standard. `ftp://ftp.rsasecurity.com/pub/pkcs/ascii/pkcs-7.asc`, Version 1.5.

[Laboratories, 2002] Laboratories, R. (2002). Pkcs #1: v2.1: Rsa cryptography standard. `ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf`.

[NIST, 2001] NIST (2001). Federal information processing standards publication 197: Specification for the advanced encryption standard (aes). National Institute of Standards and Technology. `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`.

[NIST, 2006] NIST (2006). Guidelines for media sanitization. National Institute of Standards and Technology. `http://csrc.nist.gov/publications/nistpubs/800-87/sp800-87-Final.pdf`.

[NIST, 2007] NIST (2007). Announcing the development of new hash algorithm(s) for the revision of the federal information processing standard (fips) 180-2, secure hash standard. National Institute of Standards and Technology, Commerce. `http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Jan07.pdf`.

[Novak, 2002] Novak, D. J. (2002). Governments opposition to standby counsels reply to the governments response to courts order on computer and e-mail evidence. `http://notablecases.vaed.uscourts.gov/1:01-cr-00455/docs/68092/0.pdf`, UNITED STATES OF AMERICA v. ZACARIAS MOUSSAOUI, Defendant, Criminal No. 01-455-A.

[NTI Forensics Source, 2008] NTI Forensics Source, B. S. (2008). Safeback bit stream backup software. `http://www.forensics-intl.com/safeback.html`.

[OpenSSL, 2008] OpenSSL (2008). Openssl: The open source toolkit for ssl/tls. The OpenSSL Project. `http://www.openssl.org`.

[Rivest, 1992] Rivest, R. (1992). RFC 1321: The MD5 message-digest algorithm. Status: INFORMATIONAL.

[Rivest et al., 1977] Rivest, R. L., Shamir, A., and Adelman, L. M. (1977). A METHOD FOR OBTAINING DIGITAL SIGNATURES AND PUBLIC-KEY CRYPTOSYSTEMS. Technical Report MIT/LCS/TM-82, Massachusetts Institute of Technology. `http://citeseer.ist.psu.edu/rivest78method.html`.

[Schneier, 2004] Schneier, B. (2004). Opinion: Cryptanalysis of md5 and sha: Time for a new standard. *Computerworld*. `http://www.computerworld.com/securitytopics/security/story/0,,95343,00.html`.

[Szeredi, 2008] Szeredi, M. (2008). Filesystem in usespace. `http://fuse.sourceforge.net/`.

[US Department of Defense, 2008] US Department of Defense (2008). Cac: Common access card. US Department of Defnese. `http://www.cac.mil`.

[US Treasury, 2008] US Treasury (2008). Ilook investigator. US Department of the Treasury. `http://ilook-forensics.org`.

[VMWare, 2008] VMWare (2008). Run virtual machines on your pc for free. `http://www.vmware.com/products/player/`.

[Wang et al., 2004] Wang, X., Feng, D., Lai, X., and Yu, H. (2004). Collisions functions md4, md5, haval-128 and ripemd. In *Report 2004/199*. CRYPTO 2004 Cryptology ePrint Archive. `http://eprint.iacr.org/2004/199.pdf`, rump session.

[Zimmermann, 1995] Zimmermann, P. R. (1995). *The Official PGP User's Guide*. MIT Press.

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudly Knox Library
   Naval Postgraduate School
   Monterey, California